Semi-Automated Extraction of Targeted Data from Web Pages

Estiévenart, Fabrice; Meurisse, Jean-Roch; Hainaut, Jean-Luc; Thiran, Philippe

Published in:
IEEE ICDE Workshop Proceedings

Publication date:
2006

Semi-Automated Extraction of Targeted Data from Web Pages

Fabrice Estiévenart
CETIC
Gosselies, Belgium
fe@cetic.be

Jean-Roch Meurisse
Jean-Luc Hainaut
Computer Science Institute
University of Namur
Namur, Belgium
{jrm,jlh}@info.fundp.ac.be

Philippe Thiran
Information and System Management
University of Namur
Namur, Belgium
pthiran@fundp.ac.be

Abstract

The World Wide Web can be considered an infinite source of information for both individuals and organizations. Yet, if the main standard of publication on the Web (HTML) is quite suited to human reading, its poor semantics makes it difficult for computers to process and use embedded data in a smart and automated way.

In this paper, we propose to build a bridge between HTML documents and external applications by means of so-called mapping rules. Such rules mainly record a semantic interpretation of recurring types of information in a cluster of similar Web documents and their location in those documents. Relying on these rules, HTML-embedded data can be extracted towards a more computable format. The definition of mapping rules is based on direct user input mainly for the interpretation part, and on automatic computing for the location of data in HTML tree structures. This approach is supported by a user-friendly tool called Retrozilla.

1. Introduction

Composed of Web sites interconnected by hyperlinks, the World Wide Web can be seen as a huge but chaotic source of information. Encoded in a semantically poor format (HTML), Web data are well suited to human reading when rendered in a browser but cannot easily be processed automatically by software agents. In order to provide the latter with a better access to Web content, it is necessary to build a bridge between them and HTML-embedded data. Such a bridge should tell software agents what kinds of data they can find in specific sources and where to find them precisely. Typically, this information is recorded in so-called *mapping rules*.

Our approach is depicted in Figure 1 and can be summarized as follows. (1) Given a data-intensive Web site, its pages are gathered into page clusters according to both their semantic content and their HTML structure; (2) for each cluster, recurring data of interest are given a semantic interpretation that is associated with location information to compose a mapping rule; (3) these mapping rules allow to extract HTML-embedded data towards an XML structure for further processing. Data migration [18], data integration [3] and information monitoring [17] are some of the possible exploitations of the extracted data.

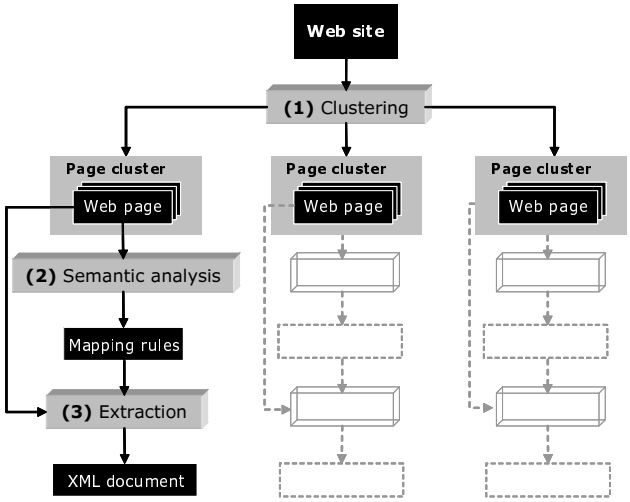


Figure 1. Overview of our approach

In this paper, we focus on the second step, i.e., building the mapping rules. To do this, a restricted set of Web documents (referred to as a *working sample* in the following) belonging to the same page cluster are rendered in a browser. A human actor, preferably aware of the semantic field associated to the page cluster, is in charge of analyzing these pages. According to his/her needs, s/he identifies, locates and gives his/her own interpretation to interesting data values, regardless of the HTML syntax. From these

inputs, mapping rules are automatically computed by a rule generator.

Retrozilla is a toolbox that implements this approach, allowing end-users to build mapping rules from a set of Web documents displayed in tabbed windows. The system is characterized by its ease of use and high flexibility. Indeed, no specific knowledge of HTML intricacies or complex mapping languages is needed to use *Retrozilla*. The user directly interacts with the Web documents in a user-friendly browser interface. As regards flexibility, this approach allows to address only the pieces of information that are of interest to the user. Furthermore, *Retrozilla*-generated mapping rules are able to process Web documents even if they show major differences in their structure and when unexpected constructs are encountered. Indeed they are built from a representative set of Web documents that most often shows all the discrepancies between the pages of the cluster.

In addition to *Retrozilla*, an extraction processor has been developed. It interprets the mapping rules in order to produce an XML document containing the target data and an XML Schema [12] document representing their data structure.

The paper is organized as follows: in Section 2, the main concepts of our approach are defined: *page cluster*, *page component* and *mapping rule*. Section 3 details the method used to build mapping rules. The extraction of data towards XML is presented in Section 4. The tools developed to support the building of mapping rules are introduced in Section 5. An overview of related work is given in Section 6. Finally, Section 7 gives some concluding remarks and plans for future work.

Examples taken from the imdb Web site (`http://www.imdb.com`), an on-line movie database will be used to illustrate our approach.

2. Main concepts of our approach

In this section, we present the major concepts needed to understand our approach. These concepts are respectively *page cluster* (Section 2.1), *page component* (Section 2.2) and *mapping rule* (Section 2.3).

2.1. Page cluster

In our approach, the pages composing a Web site are partitioned into *page clusters* [18], according to their semantic content and their layout. In other words, they display the same kind of information in a similar way. In the related literature, such a division is also called *page class* [7] or *collection* [1].

Various techniques of clustering can be found in the literature. They range from simple analysis of URLs [7], [20], to more complex criteria such as table layout [20], tags periodicity [7], keywords frequency [22] and navigational distance from the home page [7]. Most often, several techniques are used in parallel or sequentially in order to improve the accuracy of the computed clusters.

Being a field of research by itself, the page clustering issue will not be further discussed. Within the context of this paper, we only rely on a set of heuristics to identify page clusters. Thus, two pages belong to the same page cluster if they share the following intuitive features:

- they come from the same Web site (domain);
- they display instances of the same concept (e.g., two pages featuring details of a movie);
- they have a close HTML structure.

Each cluster is given a meaningful name that represents the main concept featured in its pages.

Figure 2 shows two pages of the imbd-movies cluster. For illustration purposes, their size was reduced (e.g., the list of actors was shortened).

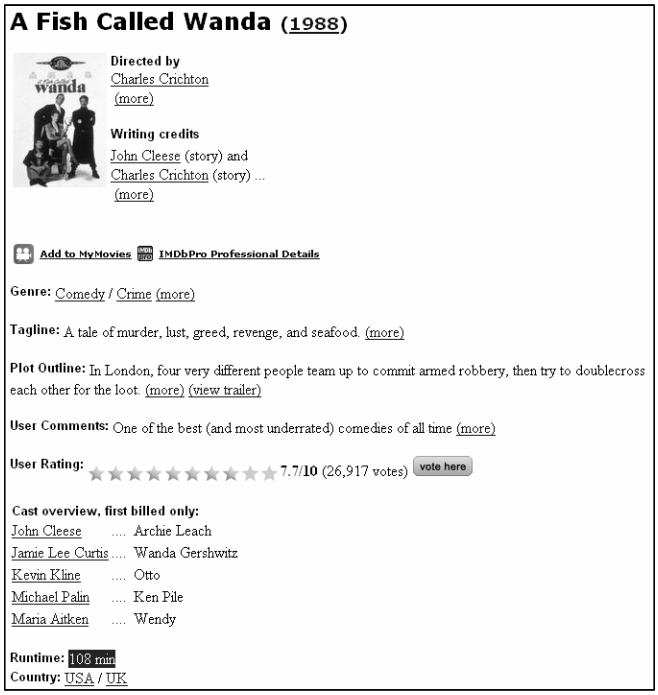
2.2. Page component

An information unit identified in a page is called a *page component*. Semantically speaking, a page component is an interesting attribute of the main concept featured in the pages of a given cluster (e.g., the runtime of a movie). Syntactically speaking, a page component is a recurring block in the pages of the cluster (e.g., HTML heading tags surrounding a character string). Each page component is given the following properties:

- A unique *name*, which identifies the page component with respect to others,
- its *optionality*, stating whether the component may be missing in some pages,
- its *multiplicity*, stating whether only one or several consecutive instances of it can appear in a page,
- its *format*, which distinguishes pure text, if the value is only composed of simple text and mixed if it is composed of both text and formatting elements,
- its *location* in the source documents.

While a page component is linked to a cluster, each of its instances in the pages of the cluster are called *component values*.

The values of the properties addressing a given page component form a tuple that we call a *mapping rule*. This tuple must provide enough information to extract instances of the component as XML elements and to give it a semantically pertinent name.



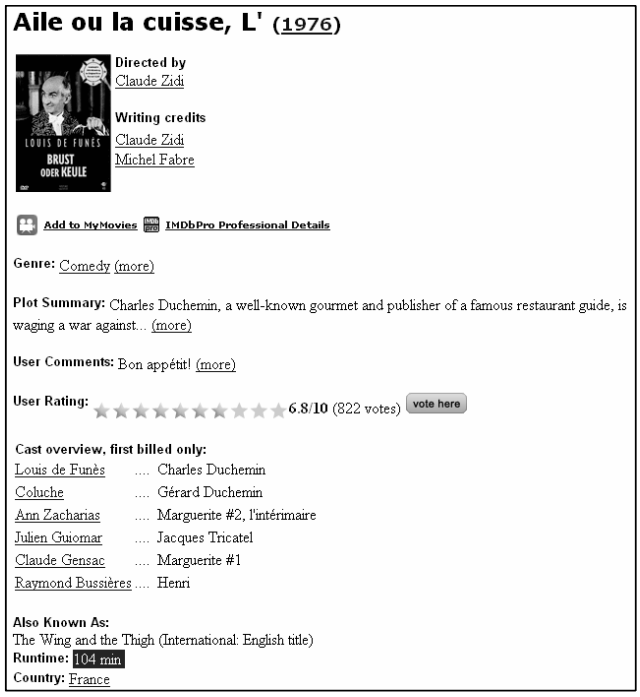


Figure 2. Two pages of the "imdb-movies" cluster

2.3. Mapping rule

A *mapping rule* (also called *extraction rule* [14] or *pattern* [3] in the related literature) is the formalization of the properties of a page component. Each mapping rule addresses exactly one page component, and, conversely, a page component can be mapped by exactly one mapping rule.

The first four properties, i.e., *name*, *optionality*, *multiplicity* and *format* are model-independent, that is they could be reused for the same purpose with non-HTML documents. They can be formalized as follows in EBNF notation:

name ::= $[a\text{–}zA\text{–}Z]([a\text{–}zA\text{–}Z] \mid [\_] \mid [0\text{–}9])*$
optionality ::= 'optional' | 'mandatory'
multiplicity ::= 'single-valued' | 'multivalued'
format ::= 'text' | 'mixed'

On the other hand, the last property, i.e., *location*, strictly depends on the underlying model and thus needs to be formalized according to it. For instance, locating a piece of information in an HTML document and in a RTF (Rich Text Format) one cannot be done the same way. Since we are dealing with DOM-compliant documents [4], an XML-related technology, namely XPath [5] has been chosen to represent the *location* property.

XPath is a language that allows to select node sets in DOM trees through node path expressions. An XPath expression can match simple leaf nodes or complex ones. It can also return multiple nodes or void results. These abilities are of high importance to us, since they can address the *optionality*, *multiplicity* and *format* properties of page components. Furthermore, predicates can be associated to XPath expressions in order to constrain or broaden their selection scope, thus enabling to produce very precise results.

On the other hand, XPath expressions always select full nodes. That feature does not allow a part only of a text node to be extracted. Consequently, the extracted data will sometimes require post processing in order to remove their noisy parts.

A sample mapping rule is given below. It addresses the runtime component (i.e., the length of the movie) in imbd-movies pages.

name : runtime
optionality : mandatory
multiplicity : single-valued
format : text
location : BODY[1]/DIV[2]/TABLE[3]/TR[1]/TD[3]/TABLE[1]/TR[6]/TD[1]/text()[1]

3. Building mapping rules

In this section, we describe the scenario through which mapping rules are built for a specific page cluster (Figure 3). This methodology is semi-automated as it requires both human intervention and automatic rules deduction.

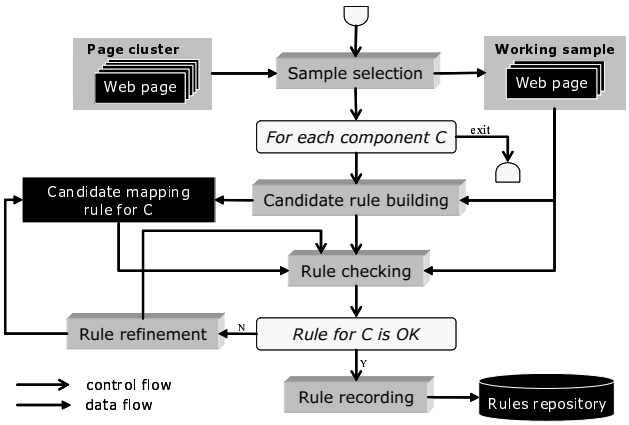


Figure 3. Mapping rules building scenario

3.1. Sample selection

A representative set of pages is selected to form a *working sample*. These pages belong to a same page cluster, but must ideally exhibit the major structural discrepancies that can be found amongst the pages of this cluster. Practice has shown that a sample of about ten randomly selected pages usually includes most of these variants. Other works [6] report that mapping rules converge after the analysis of about 5 pages.

Throughout the process, each page of the working sample is likely to enhance the quality and the accuracy of the mapping rules.

The following steps are performed for each component of interest from the user's point of view.

3.2. Candidate rule building

A *candidate rule* is a mapping rule that defines a component in (at least) one page of the working sample. Such a rule is built from the *selection* and the *interpretation* of a component value in one (randomly chosen) page of the sample. Selection and interpretation are two kinds of user-input from which a candidate mapping rule can be automatically computed.

Selection consists in pointing (and thus locating) a component value in one page of the sample. This operation leads to the automatic generation of a precise XPath expression, i.e., an XPath where each HTML element is associated with its parent-relative position, leading to the focused value. The XPath is tentative and is assigned to the *location* property of the candidate rule.

Interpretation is the process through which a semantic meaning is given to the selected component value. In our approach, the name given to a component (i.e., its semantic interpretation) is not automatically inferred [6] but rather specified by a human operator. By doing so, we focus on flexibility and precision rather than on automation.

The properties of a candidate mapping rule are deduced according to the following principles:

- *location* and *name* are respectively given by selection and interpretation.
- *optionality* and *multiplicity* are set to mandatory and single-valued respectively. Because a candidate rule first considers one page of the sample, the component is obviously considered mandatory. In this preliminary process of the scenario, the selected component value is always a single instance of the component and is therefore considered single-valued.
- *format* is set to text if the selected component value is a simple text node; otherwise, it is set to mixed .

3.3. Rule checking

The candidate rule is applied on the successive pages of the working sample to check whether it can retrieve the pertinent component values in all of them. This checking is carried out by means of visual inspection in a tabular view which displays the retrieved data in each page of the working sample. Table 1 displays the matched data for the component runtime of a 4-page working sample of the imdb-movies cluster.

Table 1. Candidate rule checking for component "runtime"

	Page URI	Component value
a.	./title/tt0095159/	108 min
b.	./title/tt0071853/	91 min
c.	./title/tt0074103/	The Wing and the Thigh (International: English title)
d.	./title/tt0102059/	-

The perfect case arises when the candidate rule matches the targeted value in every page. In such a case, the rule is recorded (Section 3.5). If, on the contrary, a mismatch occurs, then the rule must be refined (Section 3.4).

In Table 1, the candidate rule has selected the expected text node for rows a and b . However, the "min" suffix will have to be removed in order to get the proper data. Rows c and d show respectively unexpected and void values. Consequently, the rule refinement process is required.

3.4. Rule refinement

Generated from one positive example, a candidate rule is frequently too specific to locate the expected component values in all the pages of the working sample. Several situations may actually occur:

- In (at least) one page of the sample, the value matched by the candidate rule is an unwanted value (e.g., instance of another component, intrusive fragment) (Table 1, row c);
- In (at least) one page of the sample, the candidate rule cannot match any value (Table 1, row d);
- In (at least) one page of the sample, the value matched by the candidate rule is incomplete; this situation occurs when the component value is made of text only in some pages and of text and HTML tags in other pages;
- In (at least) one page of the sample, the value appears to be multivalued.

In order to solve these conflicts and build a valid mapping rule, we enter an iterative process during which the candidate rule is refined, each negative example (i.e., pages in which the candidate rule was not able to locate the expected component value) being handled one at a time.

Several strategies can be adopted, depending on the specific problem to address. We explain some of them.

Adding contextual information

As already stated, a candidate rule is as specific as possible. Sometimes, locating a component value through a position-based XPath may prove insufficient. For instance, an optional component may produce position shifts in some pages, thus leading to unwanted or void results.

In such a case, a good solution is to remove the position information where the shift occurs and to add contextual information in terms of a constant character string that always visually appears before (or after) the targeted value. The XPath is thus more flexible as regards syntax (position of tags), and the addition of contextual information ensures that component values will be reliably located.

In terms of tree structure, it consists in locating a node on the basis of its relative location with respect to another text node that always comes before (or after) it. Trees are traversed according to a Depth First Search, which is the most natural way of reading a document.

Given two Web pages (partially represented by their HTML code in Figure 4) of a working sample related to imdb-movies , we want to define a mapping rule for a component named runtime . From a selection in the first page, a candidate rule is generated. It includes the XPath of Table 2, row a. Because this first candidate rule matches a wrong

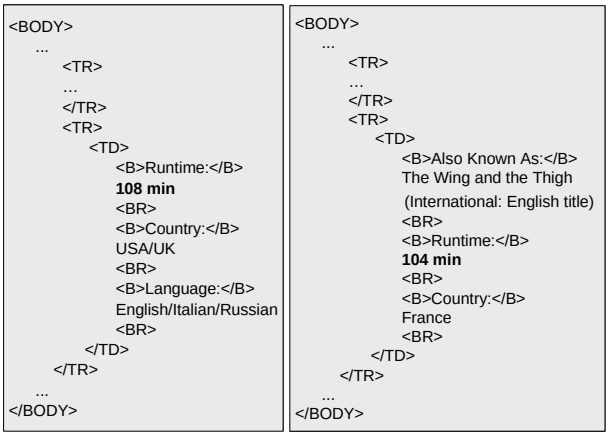


Figure 4. Using contextual information

data item in the page on the right-hand side of Figure 4 (”The Wing and the Thigh (International: English title)”), it needs to be refined by adding a contextual specification. Indeed, in both pages, the component is always preceded by the constant string ”Runtime:” The refined XPath is shown in Table 2, row b: the erroneous position predicate is replaced by a predicate searching for a specific text node in the preceding and ancestral nodes. This latest XPath expression allows the right component values to be selected in all the pages of the sample.

Table 2. Examples of valid XPath expressions

	XPath expression
a.	BODY//TR[6]/TD[1]/text()[1]
b.	BODY//TR[6]/TD[1]/text()[ancestor-or-self/preceding-sibling//text()[contains(”Runtime:”)]]
c.	BODY//TABLE[1]/TR[1]
d.	BODY//TABLE[1]/TR[position()>=1]
e.	BODY//TABLE[1]/TR[2]/TD[2]/text()
f.	BODY//TABLE[1]/TR[17]/TD[2]/text()

Table 3 shows the selected component values after the refinement of the rule related to the runtime component.

Table 3. Rule checking after rule refinement

	Page URI	Component value
a.	./title/tt0095159/	108 min
b.	./title/tt0071853/	91 min
c.	./title/tt0074103/	104 min
d.	./title/tt0102059/	84 min

Optionality, multiplicity and format properties

Because all the pages of a cluster have a similar but possibly non-identical structure, a component identified in a page can be missing in other ones. In such a case, the candidate rule selects a wrong component value (or no value at all). To handle this, the *optionality* property is set to optional.

In the preliminary phase of candidate rule building (Section 3.2), the result returned by the rule is always a single component value. However, Web pages often display consecutive pieces of information of the same type (list of comments, items, people). In HTML, these repetitive components mainly appear in the form of bulleted lists or table rows. To handle this, the position predicate associated to the repetitive tag is broadened in order to select consecutive component values. For instance, in Table 2, the XPath expressions in row c and d respectively select the first row of an HTML table and each row of the same table.

Declaring a component multivalued is also part of the rule refinement process. The repetitive tag is automatically deduced by the comparison of the XPath expressions locating the first and the last instances of the multivalued component. For example, if rows e and f in Table 2 lead to the first and the last values of a multivalued component, the repetitive element is undoubtedly <TR>.

Sometimes, the located value turns to be incomplete. In such a case, the problem lies in the fact that the expected value is composed of a single text node in some pages and of text nodes and HTML tags in other pages. To fix that, the *format* property is set to mixed.

Adding an alternative path

Adding contextual information and modifying the properties of a rule do not always give results that are satisfactory enough. Sometimes, we have still to deal with missing or wrong values.

In such a case, a component value is selected in a page where it could not be located to produce a new XPath expression that is appended to the mapping rule.

3.5. Rule recording

Once the candidate rule has been validated for the component values in all the pages of the working sample, it is recorded in a rule repository. This repository will be used by external agents, for instance by the XML extractor.

4. XML extraction

The output of the analysis process can be understood as a primitive three-level XML structure made of a root element representing the page cluster, a second level element for each page of the cluster and a leaf element for each page component. Figure 5 shows a part of the XML document generated for the imdb-movies cluster, assuming that only the runtime component has been defined.

If this three-level structure does not fit the user's view of the data, it can be transformed by iterative aggregation of the component elements into a richer tree structure that either better represents the intrinsic page structure, or meets other user requirements. For instance, in our imdb-movies cluster, the leaf components comments and rating could be embedded into a higher level component called users-opinion. In such a case, this enhanced structure is recorded in the rule repository.

The XML export is handled by a Java application. First, it uses the information contained in the rule repository to generate a data structure in the form of an XML Schema document. To be more precise, the *name* property of a mapping rule becomes the name of an XML Schema element, while the *optionality* and *multiplicity* properties are transformed into cardinality constraints in the target structure. If enhanced structure information is recorded in the rule repository, it is used to produce a nested data structure accordingly. Otherwise, the default three-level structure is used. Second, the Java application relies on the mapping rules stored in the rule repository to extract the targeted data from the HTML pages of the corresponding cluster. These data are stored into an XML document that complies with the generated XML Schema structure.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<imdb-movies>
    <imdb-movie uri="http://imdb.com/title/tt0095159/">
        <runtime>108 min</runtime>
    </imdb-movie>
    <imdb-movie uri="http://imdb.com/title/tt0071853/">
        <runtime>91 min</runtime>
    </imdb-movie>
    <imdb-movie uri="http://imdb.com/title/tt0074103/">
        <runtime>104 min</runtime>
    </imdb-movie>
    …
    <imdb-movie uri="http://imdb.com/title/tt0102059/">
        <runtime>84 min</runtime>
    </imdb-movie>
</imdb-movies>
```

Figure 5. Example of a generated XML document

5. The Retrozilla tool

This section describes *Retrozilla*, a Mozilla-based [15] prototype dedicated to the building of mapping rules.

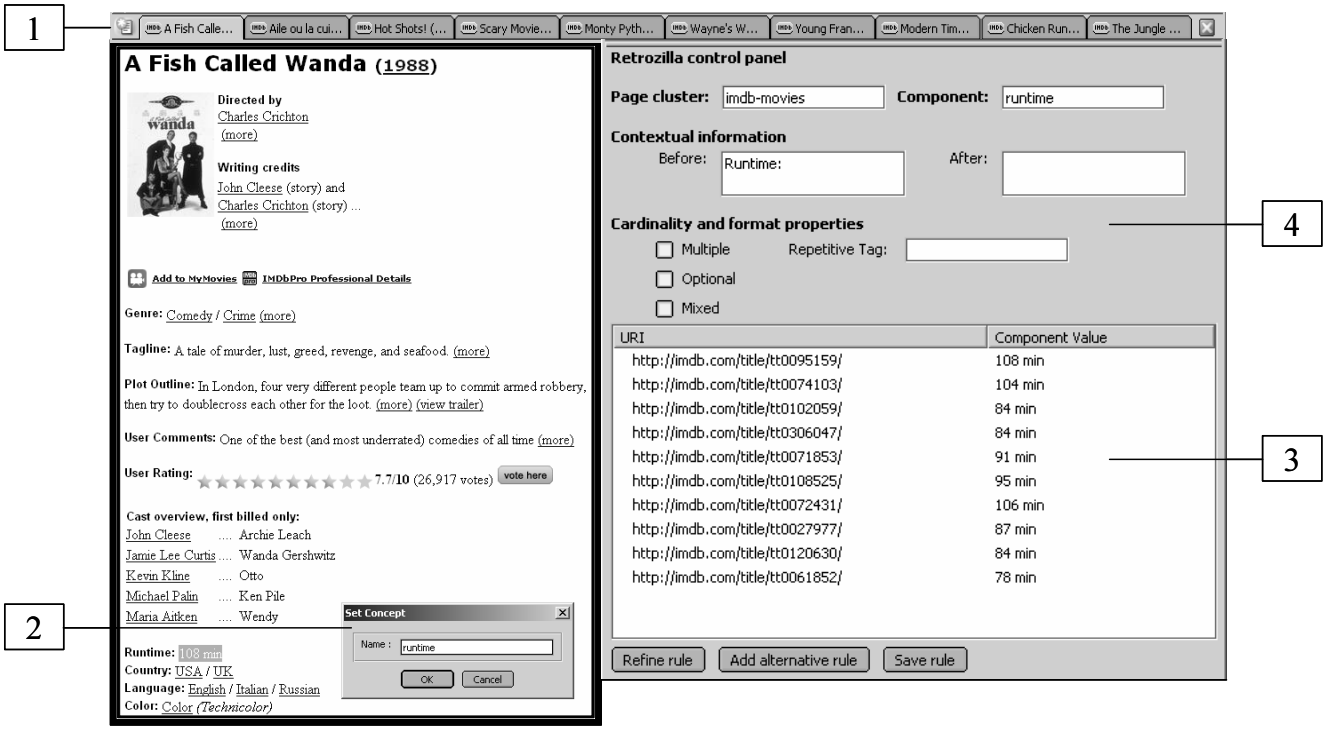


Figure 6. The Retrozilla tool

The Mozilla platform was chosen for the following reasons:

- Extensibility: Mozilla provides a cross-platform development framework that allows developers to create applications from accessible components (Javascript, XUL, XPCOM).
- DOM interpretation of Web documents: Mozilla provides an internal DOM representation of loaded HTML documents, whatever their syntactical quality. This hierarchical view of documents is necessary to locate components by means of XPath expressions.
- Tab-browsing facilities: when each page of a working sample is loaded in a tab, it is very comfortable to switch between them during the rule refinement step.
- Built-in XPath engine: Mozilla provides a built-in XPath engine that allows to select nodes in loaded documents.

5.1. The Retrozilla GUI

Developed as a plug-in for the Mozilla browser, *Retrozilla* is a toolbox that allows to perform each step of the scenario described in Section 3 in a user-friendly way. The *Retrozilla* Graphical User Interface (Figure 6) includes a Mozilla browser and a control panel.

Each page of the working sample is loaded in a tab of the Mozilla browser (Figure 6, square 1). After the selection of a component value in the browser, a dialog box allows its interpretation and the generation of a first candidate rule (Figure 6, square 2).

For rule checking, the control panel of *Retrozilla* includes a table similar to the one depicted in Table 1. The candidate mapping rule under definition can thus be easily validated by a visual inspection of the component values (Figure 6, square 3) retrieved in each page of the sample. Rule refinement and recording are also realized with the help of the control panel which receives user input and permanently displays on the fly the values matched by the mapping rule (Figure 6, square 4).

The screenshot of Figure 6 shows the definition of the runtime component and the application of this definition to the other pages of the working sample after the addition of contextual information.

5.2. The mapping rule builder

The *mapping rule builder* uses the internal DOM representation to compute the XPath leading to the value selected in the page displayed on the screen.

In addition, this component is responsible for gathering user inputs and associating them with the XPath in order to

produce a candidate rule or modify it during the refinement process.

6. Related work

Mapping rules building is seldom discussed as a standalone topic in the literature. Most often, it is regarded as part of a broader process, i.e., Web data extraction. As already mentioned in the introduction of this paper, in our view, mapping rules can be used for various other purposes.

However, since an application dedicated to the extraction of Web data towards XML has been developed too, positioning our work with respect to other Web data extraction approaches seems relevant.

Various techniques are proposed in the literature for Web data extraction: declarative languages [9], [2], wrapper induction [10], [16], deduction from ontologies [21].

By using XPath expressions to select data in Web documents, the *Retrozilla* approach is based on a technique called *HTML tree structure analysis*. Other projects also rely on this technique. Many of them, just like *Retrozilla*, have developed a customized Web browser to build data extraction modules (most often called *Web wrappers*) in a user-friendly way [3], [12], [19], [14].

Amongst them, Lixto [3] uses Elog, an expressive but complex tree query language compared to XPath. XWRAP [12] also uses tree paths and allows hierarchical structure extraction. However, XWRAP-generated mapping rules are expressed in a more procedural way (i.e., with conditional and loop structures) rather than in a flat static structure, as *Retrozilla* does. The wrapping language of W4F [19] is the HTML Extraction Language (HEL). Its particularity is the possibility to navigate the HTML tree either along the document hierarchy (like traditional XPath expressions) or along the stream of the document (i.e., its reading order). This latter original feature is useful to assign contextual information to components. SG-WRAP [14] is a visual tool that builds wrappers from a predefined schema. Associated to SG-WRAM [13], it also allows the detection of wrapper failures and automatic wrapper maintenance. If, on the one hand, these various systems seem to produce effective results, on the other hand, the use of complex built-in languages makes it difficult, or at least costly, to reuse these methods in applications other than data extraction.

Two other projects, Roadrunner [6] and EXALG [1], also rely on the HTML structure to produce wrappers, but they differ from all the other ones by proposing complete automation in wrapper construction. In these systems, complex algorithms iteratively compute a common grammar for documents of a given cluster by comparing them. Both projects plan to provide techniques to automatically label the extracted data with representative names, but so far a user intervention is still necessary to give a semantic interpretation to the extracted data. Another drawback of this kind of approach is the lack of flexibility as regards targeted data. Indeed, since the process is fully automated, there is no means of deciding which components must be extracted. All varying chunks of the HTML source code will be part of the extracted data, thus leading to documents containing data that do not interest some classes of end-users.

Finally, we suggested, in a previous paper [8], a complementary approach for Web data extraction and schema generation. In this work, the mapping between HTML and XML was realized by means of a *META file*, i.e., an XML representation of page clusters based on the source HTML structure. This approach allowed the extraction of complex multi-level XML structures from Web documents but required a better knowledge of HTML and XML models on the user's part.

In an objective and qualitative analysis of Web data extraction tools, [11] exposes some features commonly used to characterize such tools. It is worth evaluating *Retrozilla* with respect to these criteria, namely degree of automation, support for complex objects, page content, ease of use, XML output, support for non-HTML sources and resilience/adaptiveness (Table 4).

Table 4. Main features of Retrozilla

Feature	Value	Argumentation
Automation	Semi	Mapping rules are based on both user intervention and automatic computing.
Complex objects	Yes	A posteriori definition of complex components.
Page content	Data	Xpath expressions are best suited to data-oriented documents rather than text-oriented ones.
Ease of use	Easy	User intervention in a browser view; no technical skills required.
Xml output	Yes	The extraction of data towards XML is already supported.
Non-HTML	Could be	Mapping rules could be adapted to other source formats.
Resilience/ adaptiveness	No	Presently, the changes over time are not automatically detected. A set of mapping rules addresses only one page cluster.

7. Conclusion and future work

This paper discusses a tool-supported scenario to build mapping rules from a sample of HTML documents and ex-

tract the targeted data towards an XML document. Furthermore, such rules can be processed by external agents in order to facilitate their access to HTML-embedded data. Our approach is generic enough to be useful for multiple purposes. Indeed, since they provide information on the meaning, the location and the cardinality of page components, mapping rules can be used by a wide range of applications such as the migration of a static Web site towards a database, the integration of data coming from heterogeneous Web sites, the monitoring of Web data such as concurrent prices or stock rankings.

Retrozilla is a toolbox that implements this approach. Compared to similar approaches, the main features of *Retrozilla* are its ease of use and flexibility.

Assuming that user intervention is essential to provide a good interpretation of data, user-friendliness was made a major goal of our system. In the user's view, mapping rules are built regardless of both the HTML syntax and the mapping rules formalism.

Another important point was to make our system flexible. In other words, given a page cluster, different users can build different sets of mapping rules, according to their specific needs, what is generally not the case in other systems.

In addition, since the mapping rules are defined from a representative set of Web pages, most of (if not all) the discrepancies that can appear between pages are taken into account, thus giving fairly accurate results when extracting data.

At this stage, *Retrozilla* is unable to detect and repair automatically failures in the extraction process. However, according to our definition of mapping rules, we think that error recovery could be achieved by *Retrozilla* in a semi-automated way. For instance, a failure in a rule could be automatically detected when a mandatory component cannot be found in one page or when the extraction of a single-valued text component returns more than one node. When such a failure is detected, the rule should be refined manually from the negative examples.

Because XPath was chosen to select the component instances in the HTML documents, *Retrozilla* cannot extract only a part of a text node. That feature may become a real restriction when a text node contains more textual information than the component value (Table 3) or when the text node actually includes a comma-separated list of values of a multivalued component. Extra information could be added to mapping rules to handle this kind of situation. Using regular expressions would allow to finely select the component values within a text node to the detriment of user-friendliness.

Retrozilla is empirically more effective on fine-grained HTML structures (i.e., highly nested documents) rather than on poorly structured (i.e., relatively flat) documents. Indeed, components can be located more accurately when there are nested in a deeper structure.

In the *Retrozilla* approach, each component value is currently a text node, i.e., a leaf node in the HTML hierarchical structure (in case of a mixed component, the component value is a list of text nodes separated by HTML tags). The definition of complex components (i.e., components containing other components) is not handled by *Retrozilla*. Consequently, the raw structure of targeted data is completely flat. However a nested data structure can be produced, a posteriori, during the extraction step, by means of iterative aggregation of leaf elements (Section 4). This bottom-up approach proves to be flexible and well appropriate for agile Web data extraction, i.e., when only a few simple components need to be defined (for instance, the extraction of a stock value or a concurrent price). On the other hand, in a top-down approach, high-level blocks are first declared before their inner (leaf) content. In a previous work [8], we show that this approach can be optimally used in the context of Web sites re-engineering (e.g., Web data migration towards a database) or when complex data structures need to be declared. According to the exploitation of the extracted data, one approach will always be preferred to the other but we are working on the integration of both views to get a multipurpose environment.

While the current *Retrozilla* prototype has proven effective on small case studies, it still has to be tested on a larger scale. In this respect, we plan to carry out a performance study with a panel of potential users in order to assess both the usability of the prototype and the accuracy of the results.

In the near future we will also explore the opportunity to build mapping rules according to a pre-existing data structure (XML Schema, RDF, OWL). Such an improvement would allow schema reusability and sharing, and would make it easier to integrate data coming from various Web sites.

8. Acknowledgments

That research project is supported by the European Union (ERDF) and the Région Wallonne (DGTRE) under contract n° EP1A12030000062 130007.

References

[1] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 337–348, 2003.

[2] G. O. Arocena and A. O. Mendelzon. WebOQL: Restructuring documents, databases, and webs. In *Proceedings of the 14th International Conference on Data Engineering (ICDE'98)*, pages 24–33, 1998.

[3] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 119–128, 2001.

[4] W. W. W. Consortium. Document object model (dom). http://www.w3.org/DOM.

[5] W. W. W. Consortium. Xml path language (xpath). http://www.w3.org/TR/xpath.

[6] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 109–118, 2001.

[7] V. Crescenzi, G. Mecca, and P. Merialdo. Wrapping-oriented classification of web pages. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC)*, pages 1108–1112, 2002.

[8] F. Estievenart, A. François, J. Henrard, and J.-L. Hainaut. A tool-supported method to extract data and schema from web sites. In *Proceedings of the 5th International Workshop on Web Site Evolution (WSE'03)*, pages 3–11, 2003.

[9] G. Huck, P. Fankhauser, K. Aberer, and E. J. Neuhold. Jedi: Extracting and synthesizing information from the web. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (CoopIS'98)*, pages 32–43, 1998.

[10] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.

[11] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Record*, 31(2):84–93, 2002.

[12] L. Liu, C. Pu, and W. Han. Xwrap: An xml-enabled wrapper construction system for web information sources. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'00)*, pages 611–621, 2000.

[13] X. Meng, D. Hu, and C. Li. Schema-guided wrapper maintenance for web-data extraction. In *Proceedings of the 5th ACM CIKM International Workshop on Web Information and Data Management (WIDM'03)*, pages 1–8, 2003.

[14] X. Meng, H. Lu, H. Wang, and M. Gu. Data extraction from the web based on pre-defined schema. *J. Comput. Sci. Technol.*, 17(4):377–388, 2002.

[15] Mozilla. Mozilla suite - the all-in-one internet application suite.

[16] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[17] M. H. Nodine, J. Fowler, T. Ksiezyk, B. Perry, M. C. Taylor, and A. Unruh. Active information gathering in infosleuth™. *Int. J. Cooperative Inf. Syst.*, 9(1-2):3–28, 2000.

[18] F. Ricca and P. Tonella. Using clustering to support the migration from static to dynamic web pages. In *Proceedings of the 11th International Workshop on Program Comprehension (IWPC'03)*, pages 207–216, 2003.

[19] A. Sahuguet and F. Azavant. Building intelligent web applications using lightweight wrappers. *Data Knowledge Engineering*, 36(3):283–316, 2001.

[20] L. K. Shih and D. R. Karger. Using urls and table layout for web classification tasks. In *Proceedings of the 13th international conference on World Wide Web (WWW'04)*, pages 193–202, 2004.

[21] H. Snoussi, L. Magnin, and J.-Y. Nie. Towards an ontology-based web data extraction. In *Online Proceedings of Business Agents and the Semantic Web (BASeWEB'02)*, 2002.

[22] P. Tonella, F. Ricca, E. Pianta, and C. Girardi. Using keyword extraction for web site clustering. In *Proceedings of the 5th International Workshop on Web Site Evolution (WSE'03)*, pages 41–48, 2003.